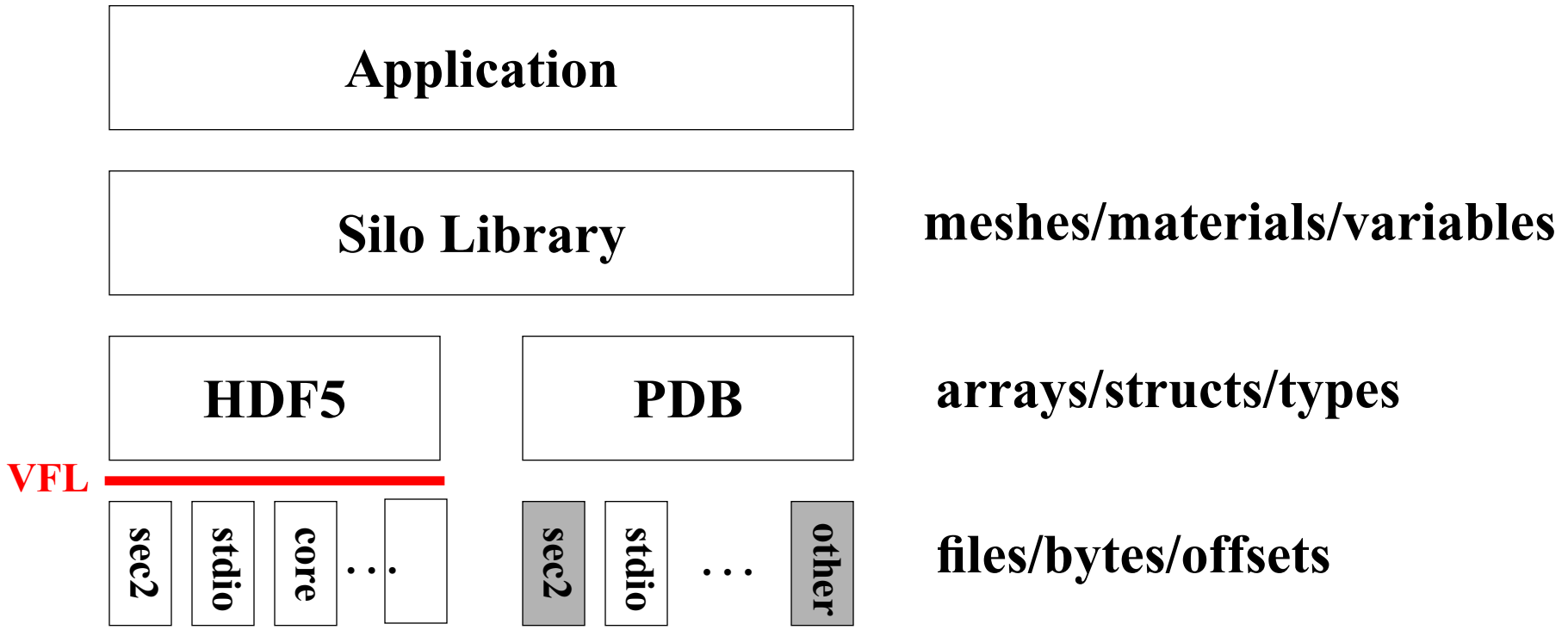
Silo/HDF5 Modifications for Dawn

Mark C. Miller

Presented at the Dawn User Forum, April 15, 2010

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
LLNL-PRES-428015

Silo Background



Benefits (= flexibility)

- *platform independent, self-describing, archiveable data*
- *random access (more true of post-processors than simulation codes)*

Drawbacks (= performance degradation)

- *metadata (data a lib writes on behalf of its caller)*
- *caller is far removed from actual disk I/O behavior/control*

Poor Man's Parallel I/O

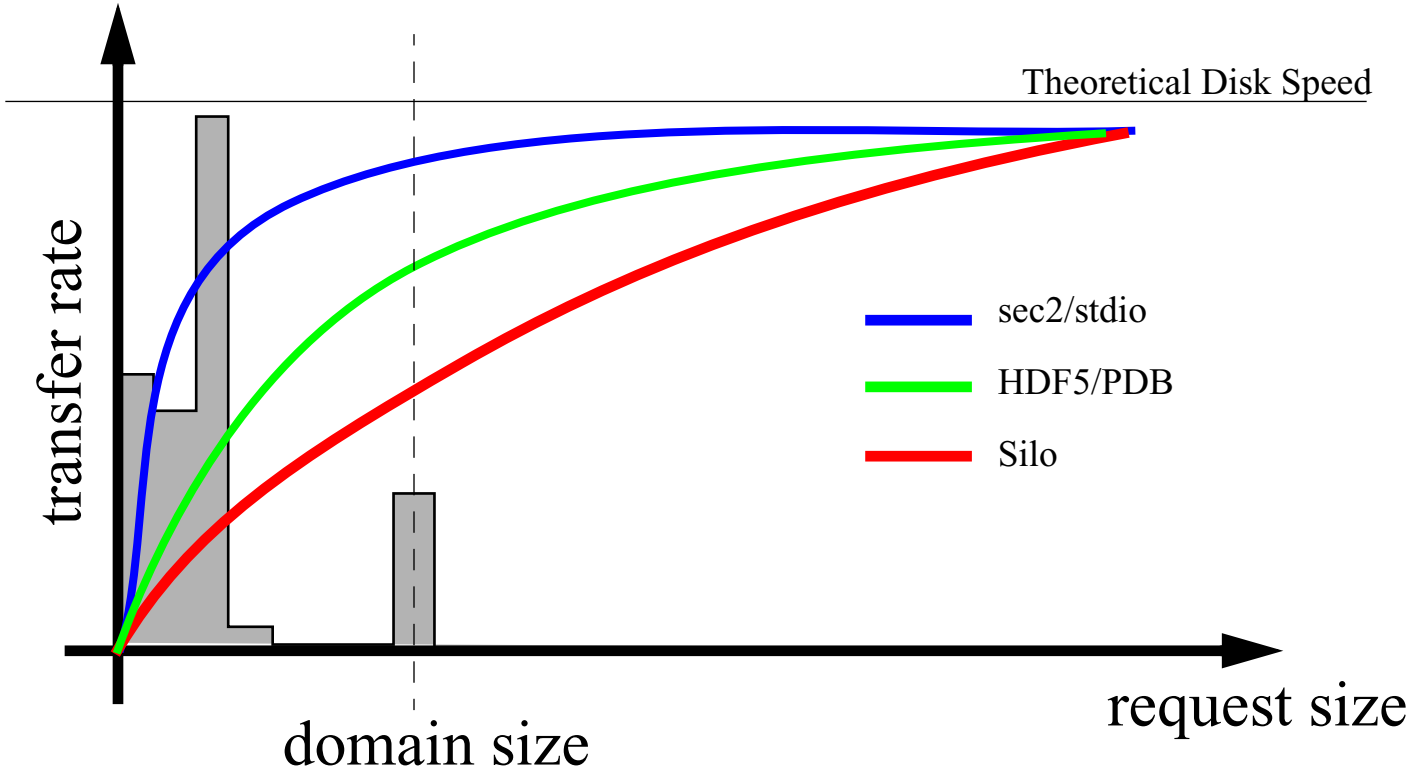
Truly concurrent, parallel I/O to a single file is problematic

- *Difficult to make perform well even for relatively simple I/O patterns.*
- *The global monolithic "whole" object is decomposed on read, re-composed on write*
- *Does not support multi-physics codes where I/O patterns are more complex*

Poor Man's Parallel I/O: Parallelism at the price of multiple files

- *Serial I/O to multiple files, simultaneously*
- *#files != #MPI-tasks*
- *Very flexible with what each MPI-task needs to do in the way of I/O*
- *Do not pay cost of "decomposing on read" and "recomposing on write"*
- *Note: Lustre can't tell the difference (almost)*

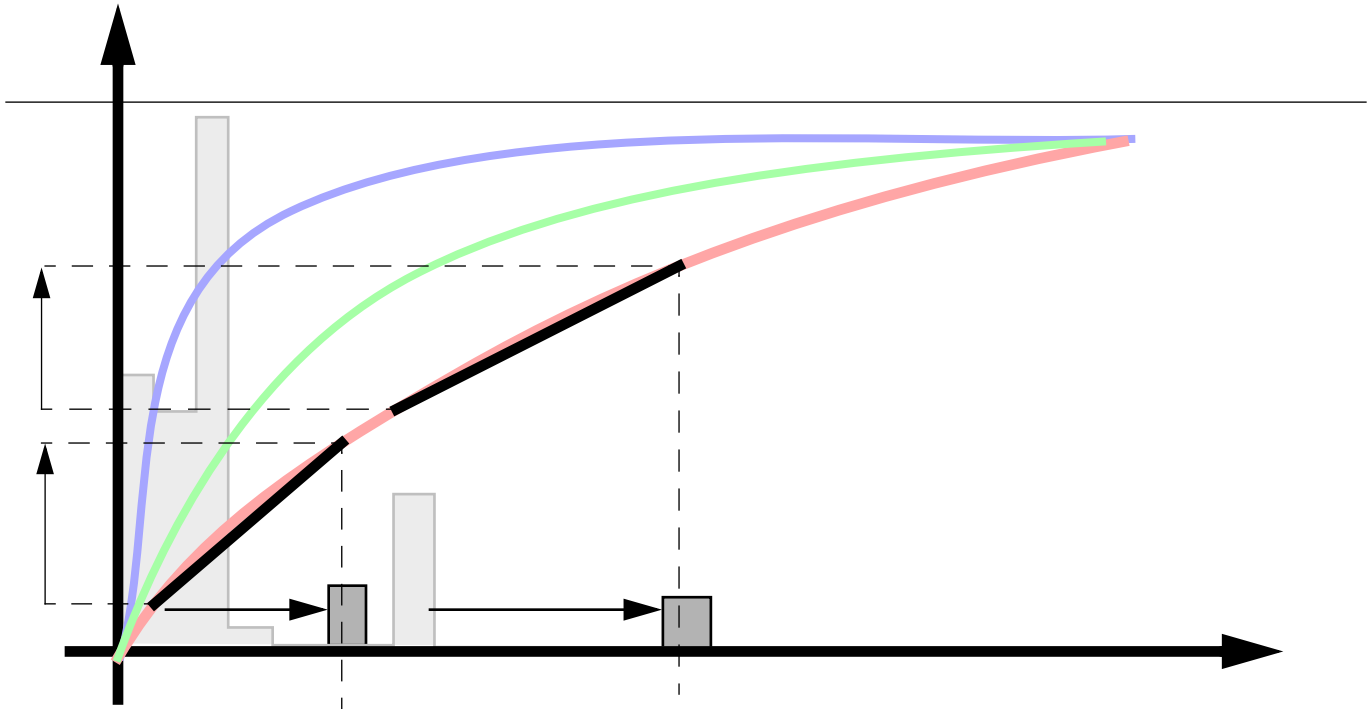
I/O Performance



Histogram of a recent Ares dump

	writes	bytes	%writes	cum.%writes	%bytes
<10^1 bytes:	48	217	20.1680	20.1680	.0001
<10^2 bytes:	41	1485	17.2268	37.3949	.0009
<10^3 bytes:	116	22474	48.7394	86.1344	.0136
<10^4 bytes:	8	30540	3.3613	89.4957	.0186
<10^5 bytes:	0	0	0	89.4957	0
<10^6 bytes:	3	1092492	1.2605	90.7563	.6655
<10^7 bytes:	22	162989412	9.2436	100.0000	99.3010

Aggregation is key to improving performance



Aggregation

- *Gather many smaller requests into fewer larger ones*
- *Need memory to do this.*
- *Try aggregating as much as possible WITHIN one MPI-task first.*
- *Failing that, start aggregating ACROSS MPI-tasks.*

Simple Aggregation Strategies

HDF5's Core VFD:

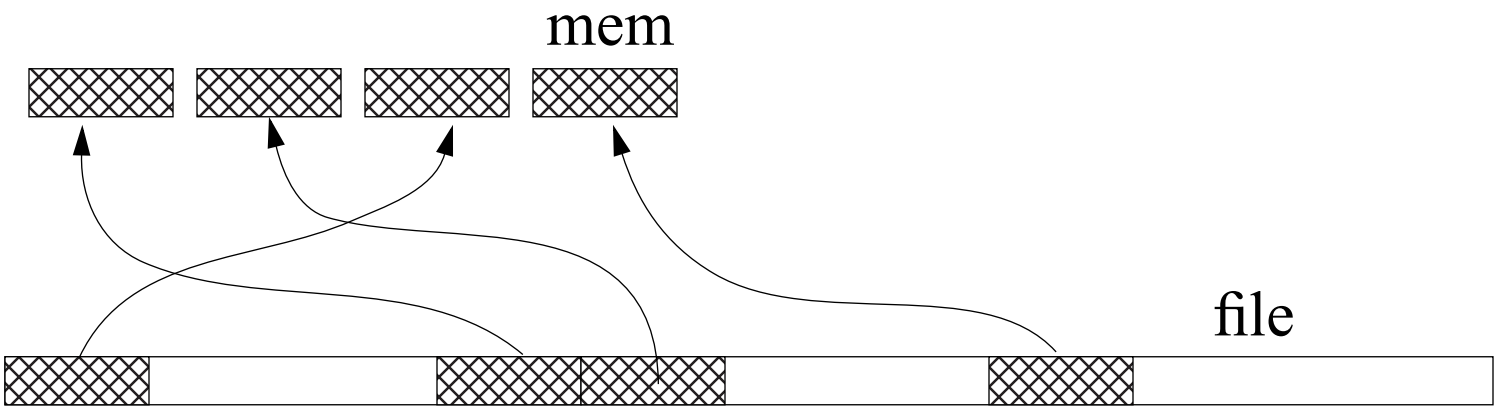
- *Stores everything to a growing buffer in memory.*
- *Writes buffer to file on close.*
- *Reads ENTIRE file to memory buffer on open.*
- *Represents upper-bound of what is possible at expense of (a lot) of memory.*
- *Only works if when code does I/O, it is dumping less than 50% of available memory.*
- *Not a good long term solution*

HDF5's Split VFD:

- *Splits data into two classes; raw and meta, writing each to its own VFD.*
- *Metadata uses core VFD, raw data uses sec2 VFD*
- *Improves performance but at price of two files on disk per one created by app.*

Silo's new Block VFD for Dawn

Breaks virtual file into blocks



Does I/O only in blocks

- *Allocates enough memory to keep N blocks in memory; uses LRU to pre-empt.*

Two Parameters set by code

- *SILO_BLOCK_SIZE (should be multiple of filesystem blocksize)*
- *SILO_BLOCK_COUNT (more is better)*

Good Values for Dawn

- *SILO_BLOCK_SIZE = (1<<20)*
- *SILO_BLOCK_COUNT=16 (16 Megabytes total)*

Other VFDs We May Write

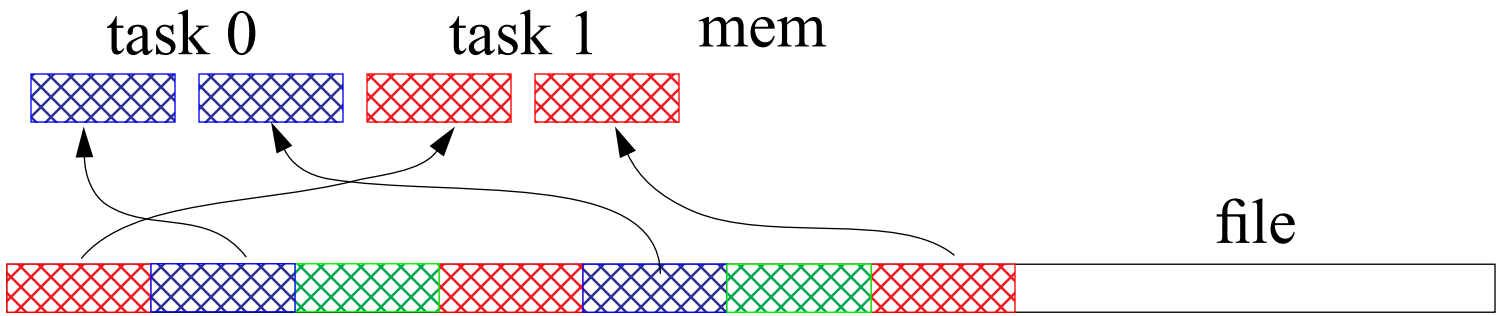
Remote-Core VFD

- *Use extra MPI-tasks just for I/O*
- *Code "writes" to memory in these extra MPI-tasks through enhanced core VFD*
- *Code goes back to compute while data drains to files from the extra MPI-tasks*
- *Should be absolute fastest as code doesn't ever wait for disk; just MPI-send(s).*

Smart-Split VFD:

- *Only one file is produced*
- *Raw data is block buffered as in new Silo VFD*
- *Metadata is kept in memory until file close, then tacked onto end of file*

Extend Block VFD to stripe across MPI-tasks



- *Let application "think" its writing to different files*
- *What if each MPI-task is writing wildly different amounts of data?*
- *May be possible to make this completely transparent to HDF5*